Versuch 1 – FreeRTOS

1. Versuchsziele

- Programmierung eines kleinen RTOS-Projektes
- Kennenlernen einiger Mechanismen von Multi-Tasking-Systemen (zyklische Tasks, Queues)

2. Wiederholungsfragen

- Was versteht man unter einem preemtiven und einem non-preemtiven Scheduler?
- Nennen Sie die Bestandteile eines prioritätsbasierten preemptiven Schedulers.
- Nennen und erläutern Sie die Zustände, die eine Task einnehmen kann!

3. Versuchsvorbereitung

- Wiederholen Sie die Vorgehensweise zur Erstellung einer Mikrocontrolleranwendung mit MPLAB X und dem Compiler C18.
- Verschaffen Sie sich einen Überblick über die Nutzung des Kernels FreeRTOS und den Microchip PICmicro (PIC18) RTOS Port.
- Testen Sie das Programmbeispiel 'Blinky' aus der Vorlesung. Legen Sie dazu ein entsprechendes Projekt unter MPLAB X an.

4. Aufgaben

a) Modifizieren Sie das Projekt 'Blinky' derart, dass jede Task in einer eigene Quelldatei gespeichert wird. Programmieren Sie dazu auch eine Haeder-Datei 'task_system.h' welche die Prototypen der Tasks und Deklarationen der xTaskHandle (Schlüsselwort extern) enthält.

b) Programmieren Sie eine weitere Task (task_adc.c), welche zyklisch fünf mal pro Sekunde einen Analog-Wert vom Eingang RA0 ließt und diesen im LCD-Display anzeigt.
Binden Sie dazu die zwei vorkompilierten Objektdateien lcd.o und ra0.o in Ihr Projekt ein, welche Ihnen Funktionen zur Nutzung des AD-Wandlers und der LCD-Anzeige bereitstellen (siehe lcd.h und ra0.h).
Zur Konvertierung des Binärwertes vom AD-Wandler in ASCII-Darstellung bietet sich die Nutzung der Funktion *void **itoa**(static int,static char *);* (stdlib.h) sowie zur Ermittlung der Stringlänge die Nutzung der Funktion *size_t **strlen**(auto const char *s);* (string.h) an.

c) Programmieren Sie für die Anzeige auf dem LCD eine getrennte Task (task_adc.c). Die Übergabe des Anzeige-Strings soll mittels einer Queue erfolgen. Programmieren Sie die Deklaration der Queue-Handle-Variable (Schlüsselwort extern) in eine eigene Header-Datei (queue_system.h).

Versuch 2 – UML und Objektorientierung

1. Versuchsziele

- Kennenlernen eines UML-basierten Entwurfsprozesses mit Codegenerierung am Beispiel der Hardware myAVR-MK2 und der Entwicklungs-Software SiSy3 AVR.

2. Wiederholungsfragen

- Nennen Sie Diagramm-Typen der UML für die Darstellung des statischen und dynamischen Verhaltens eines Systems.
- Erläutern Sie die Assoziationen Generalisierung/Spezialisierung und Aggregation.

3. Versuchsvorbereitung

- Wiederholen Sie die Themen Klassen und Vererbung unter C++.
- Verschaffen Sie sich einen Überblick über das Tool SiSy AVR mit Hilfe der Demo-Version von der Internet-Seite des Herstellers.

4. Aufgaben

a) Legen Sie auf Laufwerk D: eine Verzeichnis ***sisy*** mit zwei Unterverzeichnissen ***projekte*** und ***archive*** an. Verbinden Sie die Hardware mit dem USB-Kabel und starten Sie die Software SiSy3 AVR. Legen Sie ein neues Projekt an. Wählen Sie das AVR-Vorgehensmodell. Als Programmer ist mySmartUSB MK2 (COM5) auszuwählen, Der Quarztakt beträgt 3,6864 MHz. Im Dialog Diagrammvorlagen ist 'keine Vorlagen verwenden' auszuwählen.

Ziehen Sie das Symbol Klassendiagramm auf Ihr Arbeitsblatt. Im Dialog Klassendiagramm muß ein Name eingegeben werden. Die Optionen 'Fast-UML' sowie auf dem Tab 'Extras (AVR)' 'myAVR Controllcenter nach Erstellung starten' sind zu deaktivieren.

Über den Menüpunkt 'Nach unten (öffnen)' im Kontext-Menü des Klassendiagramms gelangt man in den Editier-Modus. Im Dialog Diagrammvorlagen ist wieder 'keine Vorlagen verwenden' auszuwählen.

Ziehen Sie eine Klasse auf das Arbeitsblatt und geben Sie ihr den Namen 'Controller'. Fügen Sie in die Klasse zwei Operationen (Zugriff: public) mit den Namen 'main' und 'run' ein.

Fügen Sie als nächstes ein Objekt ein, welches als Instanz Ihrer zentralen Anwendungsklasse genutzt wird und vergeben Sie einen geeigneten Namen. Selektieren Sie die Klasse 'Controller' und ziehen Sie mit der Maus eine Verbindung vom roten Verteiler zu Ihrem Objekt. Es wird eine Assoziation mit der Beschriftung <<instanceOf>> erstellt.

Über das Aktionsmenü kann das Projekt jetzt erzeugt und kompiliert werden (2. Erstellen). Schauen Sie sich in Ihrem Projekt-Ordner die erstellten cpp- und h-Quelldateien an.

b) Nutzen Sie für die nächsten Schritte das Benutzerhandbuch SiSy.

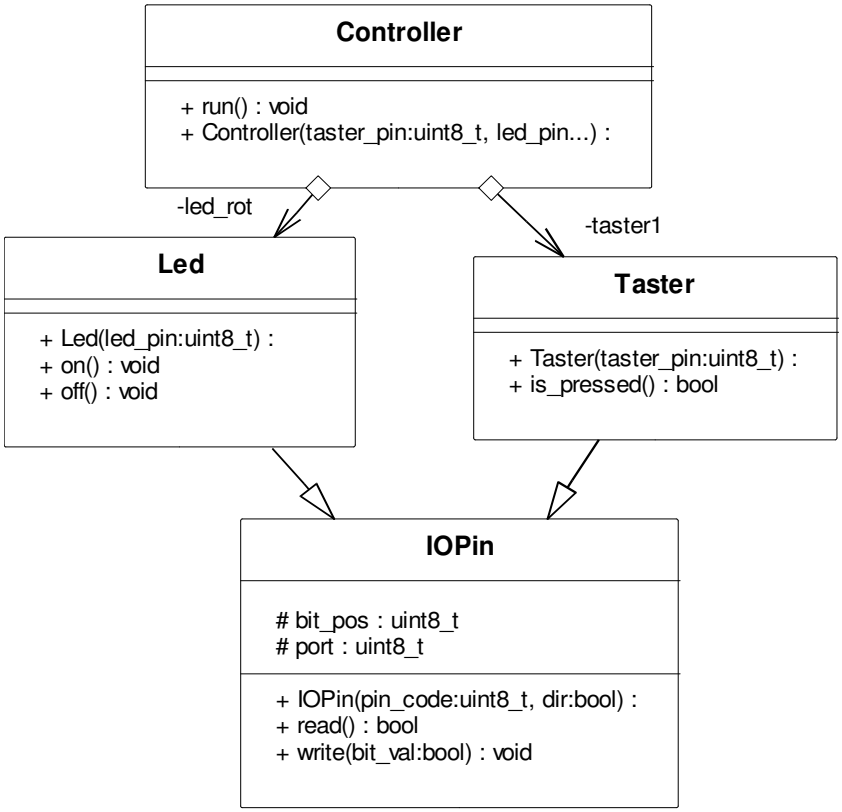
Entwerfen Sie die Systemstruktur für eine Software auf dem Mikrocontroller, bei der durch Drücken eines Tasters eine LED eingeschaltet wird (Seite 88).

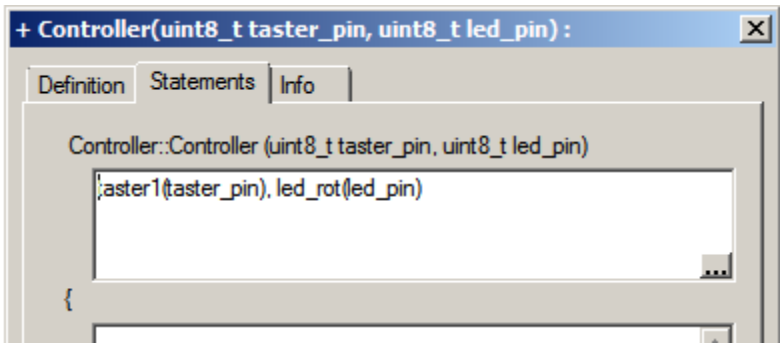
Danach ist das Systemverhalten der Klassen-Methoden zu programmieren.

Testen Sie das erstellte Mikrocontroller-Programm.

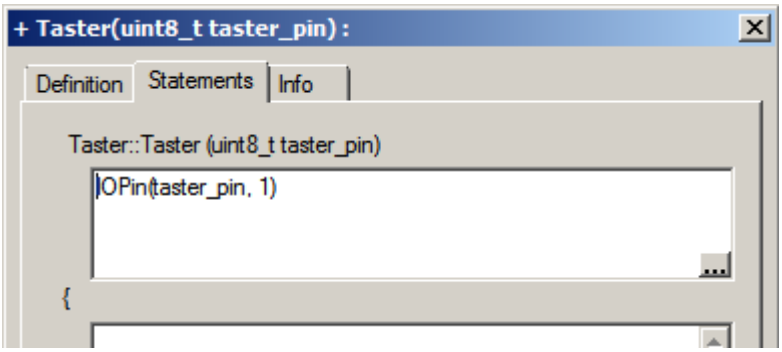
c) Die objektorientierte Struktur der Software soll jetzt verbessert werden. Es soll dazu eine Klasse IOPin programmiert werden, welche für die Verwaltung der Pin-Information zuständig ist und Ihre Eigenschaften an die Klassen Taster und Led vererbt. Das Attribut port der Klasse IOPin speichert die Port-Information (B = 1, C = 2, D = 3) des verwendeten I/O-Pins, das Attribut bit_pos enthält für den verwendeten Port-Pin (0..7) an der entsprechenden Bitstelle eine 1.

Weiter soll die Initialisierung der Objekte (app, led_rot und taster_1) über parametrisierte Konstruktoren erfolgen, so dass keine Init-Methoden verwendet werden.


Controller
+ run() : void
+ Controller(taster_pin:uint8_t, led_pin...) :
-led_rot
-taster1
Led
+ Led(led_pin:uint8_t) :
+ on() : void
+ off() : void
Taster
+ Taster(taster_pin:uint8_t) :
+ is_pressed() : bool
IOPin
bit_pos : uint8_t
port : uint8_t
+ IOPin(pin_code:uint8_t, dir:bool) :
+ read() : bool
+ write(bit_val:bool) : void


+ Controller(uint8_t taster_pin, uint8_t led_pin) :
Definition
Statements
Info
Controller::Controller (uint8_t taster_pin, uint8_t led_pin)
;aster1(taster_pin), led_rot(led_pin)
{

+ Led(uint8_t led_pin) :
Definition
Statements
Info
Led::Led (uint8_t led_pin)
IOPin(led_pin, 0)
{
// Led ausschalten
off();




```
IOPin::IOPin(uint8_t pin_code, bool dir)
{
    port = (pin_code>>4)+dir*8;
    bit_pos = (1<<(0x0F & pin_code));

    switch (port)
    {
      // DDR-Bit setzen für Eingang
      case 1: DDRB = DDRB | bit_pos; break;
      case 2: DDRC = DDRC | bit_pos; break;
      case 3: DDRD = DDRD | bit_pos; break;
      // DDR-Bit löschen für Ausgang
      // PORT-Bit setzen für Pullup
      case  9: DDRB = DDRB & (~bit_pos);
               PORTB = PORTB | bit_pos;
               break;
      case 10: DDRC = DDRC & (~bit_pos);
               PORTC = PORTC | bit_pos;
               break;
      case 11: DDRD = DDRD & (~bit_pos);
               PORTD = PORTD | bit_pos;
               break;
    }
    port = pin_code>>4;
return ;
}
```

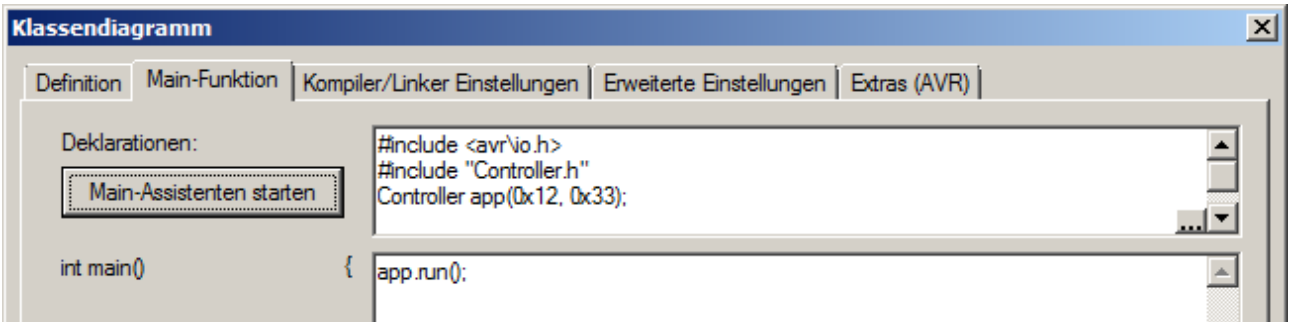
```
void IOPin::write(bool bit_val)
{
  unsigned char val_port = port+bit_val*8;

  switch (val_port)
  {
    // PORT-Bit löschen
    case 1: PORTB = PORTB &(~bit_pos); break;
    case 2: PORTC = PORTC &(~bit_pos); break;
    case 3: PORTD = PORTD &(~bit_pos); break;
    // PORT-Bit setzen
    case  9: PORTB = PORTB | bit_pos; break;
    case 10: PORTC = PORTC | bit_pos; break;
    case 11: PORTD = PORTD | bit_pos; break;
  }
}

bool IOPin::read()
{
  bool ret_val = 0;
  unsigned char mask = 0;

  switch (port)
  {
  case 1: mask = PINB & bit_pos; break;
  case 2: mask = PINC & bit_pos; break;
  case 3: mask = PIND & bit_pos; break;
  }
  if (mask != 0) ret_val = 1;

return ret_val;
}
```



Versuch 3 – Statische Codeanalyse und Moduldiagramme (C)

1. Versuchsziele

- Kennlernen und Anwenden der Tools **Splint**, **PC-lint** und **Cppcheck** in Mikrocontrollerprojekten mit dem Compiler MPLAB C18
- Erstellung einer einfachen Anwendung auf der Basis eines Moduldiagramms

2. Wiederholungsfragen

- Wodurch unterscheiden sich statische und dynamische Tests?
- Nennen Sie Fehlerarten, die mit statischer Codeanalyse gefunden werden können.
- Welche Diagrammelemente werden in Moduldiagrammen verwendet?

3. Versuchsvorbereitung

- Wiederholen Sie die Vorgehensweise zur Erstellung einer Mikrocontrolleranwendung mit MPLAB X und dem Compiler C18.
- Verschaffen Sie sich einen Überblick über die Tool Splint.
- Machen Sie sich mit der Bedienung des Modellierungs-Tools BOUML vertraut. Nutzen Sie dazu die Freeware-Version und das C++ Beispiel-Video.

4. Aufgaben

a) Starten Sie als erstes den Kommandozeileninterpreter **cmd** und kontrollieren Sie, dass die Aufrufpfade für die Ausführung der Tools Splint und PC-lint in der Umgebungsvariable PATH vorhanden sind (C:\splint-3.1.2\bin, C:\lint). Geben Sie dazu an der Kommandozeile den Befehl path ohne Parameter ein.

Bei Bedarf kann die Umgebungsvariable PATH unter Windows in den Systemeigenschaften geändert werden (sysdm.cpl).

b) Starten Sie MPLAB X und legen Sie auf Laufwerk D: eine neues Projekt für den Mikrocontroller PIC18F4520 an. Kontrollieren Sie, dass in den Projekteinstellungen für den Compiler die Option 'Enable integer promotions' aktiviert ist.

Codieren Sie das Beispiel aus der Vorlesung.

```
/* Bit-Muster für 'Start', inverses Muster für 'Stop' */
#define START 0x55          /* 0101.0101 */
#define STOP ~START         /* Bitweise Invertierung */

void main(void)
{
  char temp, temp2;
  unsigned char mask = START;
  for(;;)
  {
    if ( mask == STOP )
      temp = 'P';
    else
      temp = 'A';

    temp2 = temp;
    mask = ~mask;
  }
}
```

Speichern Sie eine Optionsdatei '.splintrc' mit folgendem Inhalt in Ihrem Projektverzeichnis.

```
# Mode-Selector-Flag
#-weak
```

```
#-standard
-checks
#-strict

# Formatierung der Ausgabe
#-showfunc
#-showcol
-linelen 60

# Unterdrückung einer Warnung, wenn main void zurückliefert (Standard int)
-maintype

#Test auf Speicherzugriffsverletzung einschalten
+bounds

#wenn logische Bedingung int-Typ keine Warnung (z.B. while(1))
-predboolint

# Unterdrückung von Nicht-ANSI-C-Schlüsselworten
-Drom=
-Dram=
-Dnear=
-Dfar=

# Include-Pfad für Header-Dateien
-IC:\Programme\Microchip\mplabc18\v3.47\h

# Ausschalten von PIC-Header-Meldungen
-declundef

# Test auf reservierte ISO-Bezeichner ausschalten
-isoreserved
```

Simulieren Sie das Programm und testen Sie den Quellcode mit Splint, PC-lint und Cppcheck. Wechseln Sie dazu im Kommandozeileninterpreter in Ihren Projekt-Pfad (Kommando **cd**).

Z.B. Projektpfad: 'D:\pic\', Quelldatei 'lint_tst.c'

```
D:\pic>splint lint_tst.c > lint.txt
D:\pic>lin lint_tst.c
```

Es empfiehlt sich, die Ausgabe-Dateien lint.txt und _LINT.TMP in jeweils einem Fenster in MPLAB X anzuzeigen.

Korrigieren Sie das Programm.

c) Ändern Sie den Vergleich in der if-Struktur in eine Zuweisung und wiederholen Sie die statischen Tests.

d) Löschen Sie den else-Zweig und wiederholen Sie die statischen Tests.

e) Führen Sie Tests auf Array-Index-Fehler durch, z.B. mit folgendem Beispiel.

```
void main(void)
{
  int lauf;
  char feld[10];

  feld[20] = 'A';

  for (lauf = 0; lauf <= 10; lauf++)
```

```
    feld[lauf] = 'A';

  for(;;){ }
}
```

f) Entwerfen Sie mit dem Tool BOUML eine Projekt-Architektur mit den Modulen Controller, Taster und Led (siehe Vorlesung). Es soll bei Drücken der Taste am Port-Pin RB0 eine LED am Port-Pin RC0 eingeschaltet werden.

Öffnen Sie als erstes nach dem Start von BOUML das Standard-C-Projekt 'C.prj' aus dem Verzeichnis 'C:\Program Files\Bouml\C' und speichern Sie dieses unter einem neuen Namen auf dem Laufwerk D: ab.

Im Projekt-Browser wird zunächst ein leeres 'package' mit Ihrem Projekt-Namen angezeigt. Legen Sie darunter ein 'class view' Objekt und ein 'class diagram' Objekt an. Im Klassen-Diagramm können Sie nun die Systemstruktur (Module Controller, Led, Taster, p18f4520) entwerfen.

Legen Sie nun auf Laufwerk D: mit MPLAB X ein neues Projekt an.

Öffnen Sie in BOUML durch Doppelklick auf das 'package' Objekt den Package-Dialog. Geben Sie auf dem Tab 'C++' in den Eingabefeldern 'header directory' und 'sources directory' den Datei-Pfad für die Quelldateien in Ihrem MPLAB X Projekt ein.

Zur Code-Generierung sind zwei Aktivitäten erforderlich. Als erstes ist im Kontext-Menü des 'class view' Objektes die Aktion 'Tool\Deploy classes' auszuführen und danach im Kontext-Menü des 'package' die Aktion 'Generate\C'.

Binden Sie die erzeugten Quelldateien in ein MPLAB X Projekt ein und testen Sie dieses auf der PIC18F4520-Hardware.

Versuch 4 – In-Circuit-Emulation

1. Versuchsziele

- Anwendung eines ICE bei Softwaretest und Fehlersuche am Beispiel des PICE-MC der Firma Phyton

2. Wiederholungsfragen

- Nennen Sie Tools für den dynamischen Softwaretest in Mikrocontrollersystemen.
- Welche Debugger-Funktionen bietet ein In-Circuit-Emulator?
- Welche Vor- und Nachteile hat ein In-Circuit-Emulator gegenüber einem In-Circuit-Debugger?
- Welche unterschiedlichen Möglichkeiten gibt es, den Emulationsprozessor für einen In-Circuit-Emulator zu realisieren?

3. Versuchsvorbereitung

- Wiederholen Sie die Vorgehensweise zur Erstellung einer Mikrocontrolleranwendung mit MPLAB X und dem Compiler C18.

4. Aufgaben

a) Stecken Sie als erstes den Emulator auf den Targetadapter auf der Zielhardware und verbinden Sie den ICE mit dem PC und der Versorgungsspannung (**3,3V**). Verbinden Sie erst danach die Target-Leiterplatte mit der Versorgungsspannug. Starten Sie nun die Software PICE-MC In-Circuit Emulator. Verschaffen Sie sich als erstes einen Überblick über die Möglichkeiten der Hardware-Konfiguration (Menü: Configure/Hardware configuration...). Wählen Sie den Chip PIC18F4520 und als Clock Generator die Option External: HS.

b) Starten Sie nun MPLAB X und legen Sie ein kleines Projekt auf LW D: an, welches bei Betätigung des externen Tasters (RB0) die LED (RA0) ein- oder ausschaltet. Laden Sie das Programm in den Emulator (Menü File\load program for debugging... , File format: COFF) und testen Sie es im Einzelschritt und Run-Modus. Legen Sie zwei globale Variablen vom Typ volatile int an, welche jeweils im if- bzw. else-Zweig inkrementiert werden.
Nach jeder Softwareänderung in MPLAB ist im Debugger die Software mit Re-load neu zu laden. Boabachten Sie die Änderung der Variablen zur Programmlaufzeit im Watch-Fenster.

c) Konfigurieren und Testen Sie einen Breakpoint bei schreibendem Zugriff auf eine der Variablen.

d) Entfernen Sie den Daten-Breakpoint wieder und konfigurieren Sie einen komplexen Breakpoint, welcher die Programmausführung nur stoppt, wenn ein bestimmter Wert in die Variable geschrieben wird.

e) Öffnen Sie das Tracer-Fenster und verschaffen Sie sich einen Überblick über die Aufzeichnung.

f) Entfernen Sie die Tasterabfrage aus der Endlos-Schleife in der main-Funktion und Programmieren Sie einen Interrrupt-Funktion für das Taster-Signal an RB0.

```
void init_ext_int(void)
{
  RCONbits.IPEN = 1; // enable interrupt priority
  INTCONbits.GIEH = 1; // enable high priority interrupts

  INTCONbits.INT0IE = 1; // Enables the INT0 external interrupt
                         // INT0 ist immer high priority
  INTCON2bits.INTEDG0 = 0;  // 0 = Interrupt on falling edge
}

/*****************************************************************/
#pragma interrupt high_isr
void high_isr (void)
```

```
{
  LATAbits.LATA0 = ~LATAbits.LATA0;
  INTCONbits.INT0IF = 0;
}

/*****************************************************************/
// Interrupt-Vektor hohe Priorität
/*****************************************************************/
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
_asm GOTO high_isr _endasm
}

#pragma code
```

Testen Sie, ob Taster-Prellen auftritt und messen Sie die Zeitverzögerung zwischen den Prell-Ereignissen mit Hilfe der Trace-Aufzeichnung. Setzen Sie dazu auf die zwei Befehlszeilen in der Interrupt-Funktion je einen 'Toggle Trace Start Trigger' und einen 'Toggle Trace Stop Trigger'.

g) Schließen Sie zum Beenden zunächst die Debugger-Software. Entfernen Sie dann als erstes die Versorgungsspannung an der Target-Leiterplatte und danach erst am ICE.

Versuch 5 – Logikanalyse

1. Versuchsziele

- Anwendung eines Low-Cost-Logikanalysators zur Verifikation von Timing und Funktionsweise von typischer Mikrocontroller-Peripherie
- Timing- und Zustandsanalyse an einer parallelen Schnittstelle zu einem Punkt-Matrix-LCD
- Timinganalyse an der seriellen I²C-Schnittstelle

2. Wiederholungsfragen

- Nennen und erläutern Sie die zwei Grundbetriebsarten eines Logikanalysators!
- Nennen Sie die drei wichtigsten Kriterien, durch welche die Leistungsfähigkeit eines Logikanalysators charakterisiert werden kann!

3. Versuchsvorbereitung

- Wiederholen Sie die Vorgehensweise zur Erstellung einer Mikrocontrolleranwendung mit MPLAB X und dem Compiler C18.
- Machen Sie sich vertraut mit der Ansteuerung eines Punkt-Matrix-LCD sowie von IIC-Slave-Bausteinen durch einen PIC-Mikrocontroller (siehe u.a. Arbeitsmaterial und Praktikumsaufgaben zur Vorlesung Mikrocontrollerpraxis)

4. Aufgaben

a) Starten Sie die Software LogicPort. In der Hilfe finden Sie unter Anhänge die Anschlussbelegung des Logikanalysators. Verbinden Sie die Logikanalysator-Kanäle mit dem SUT entsprechend folgender Tabelle.

Logikanalysator-Kanäle	**SUT**
D0 - D7	D0 – D7 am LCD-Modul
D8, D9, D10	E, RW, RS am LCD-Modul
CLK1, CLK2	E, RS am LCD-Modul
D11, D12	SDA, SCL am I²C-Peripherie-Modul

b) Programmieren Sie eine Testanwendung auf der Basis der gegebenen Funktionen des Praktikumsversuchs 'LCD-Ansteuerung mit paralleler Schnittstelle' zur VL Mikrocontrollerpraxis.

Nutzen Sie zur Initialisierung des LCD folgende Funktion.

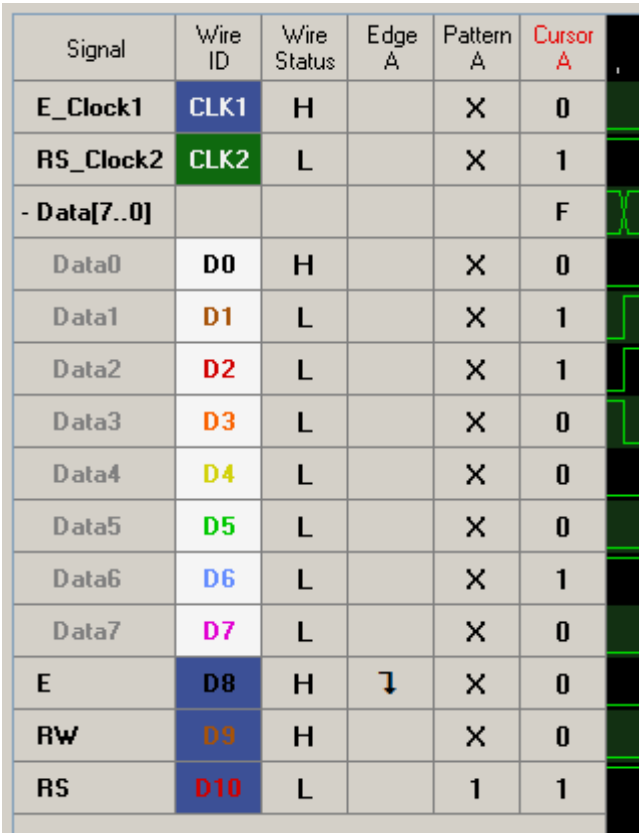
```
void init_lcd(void)
{
  // Reset Output-Value on Port E and Port D
  LATE = 0;
  LATD = 0;
  // Port E = Digital-I/O
  ADCON1bits.PCFG3 = 1;
  ADCON1bits.PCFG2 = 0;
  ADCON1bits.PCFG1 = 1;
  ADCON1bits.PCFG0 = 0;
  TRISEbits.TRISE0 = 0;
  TRISEbits.TRISE1 = 0;
  TRISEbits.TRISE2 = 0;
  wait_for_busy();
  // Set: 8 bit Interface, 2 line, 5x7 dots
  displ_write(instr_reg, 0b00111000);
  // Display On, Cursor Off, Blinking Off
  displ_write(instr_reg, 0b00001100);
  // Display Clear
```

```
  displ_write(instr_reg, 0b00000001);
}
```

Geben Sie in der Endlos-Schleife der main-Funktion jeweils ein Zeichen in jeder Zeile aus.

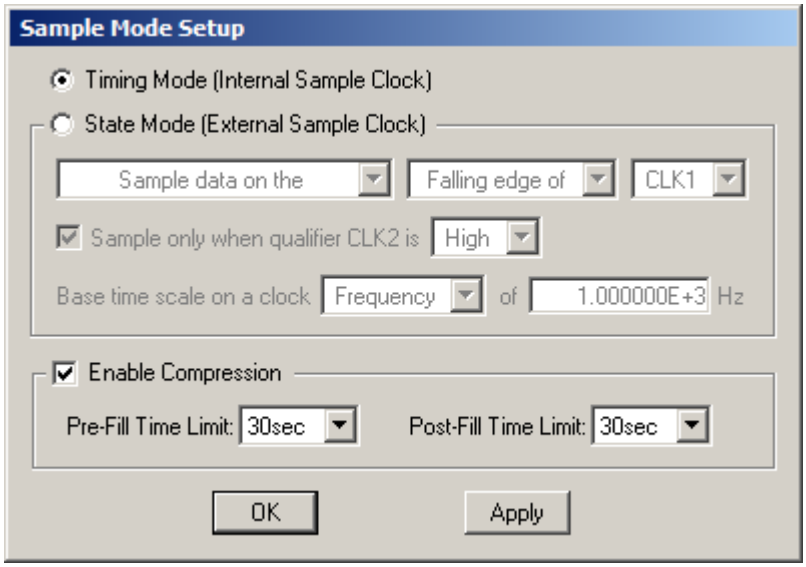
```
displ_write(instr_reg, 0x80);
displ_write(data_reg, 'H');
displ_write(instr_reg, 0xC0);
displ_write(data_reg, 'F');
```

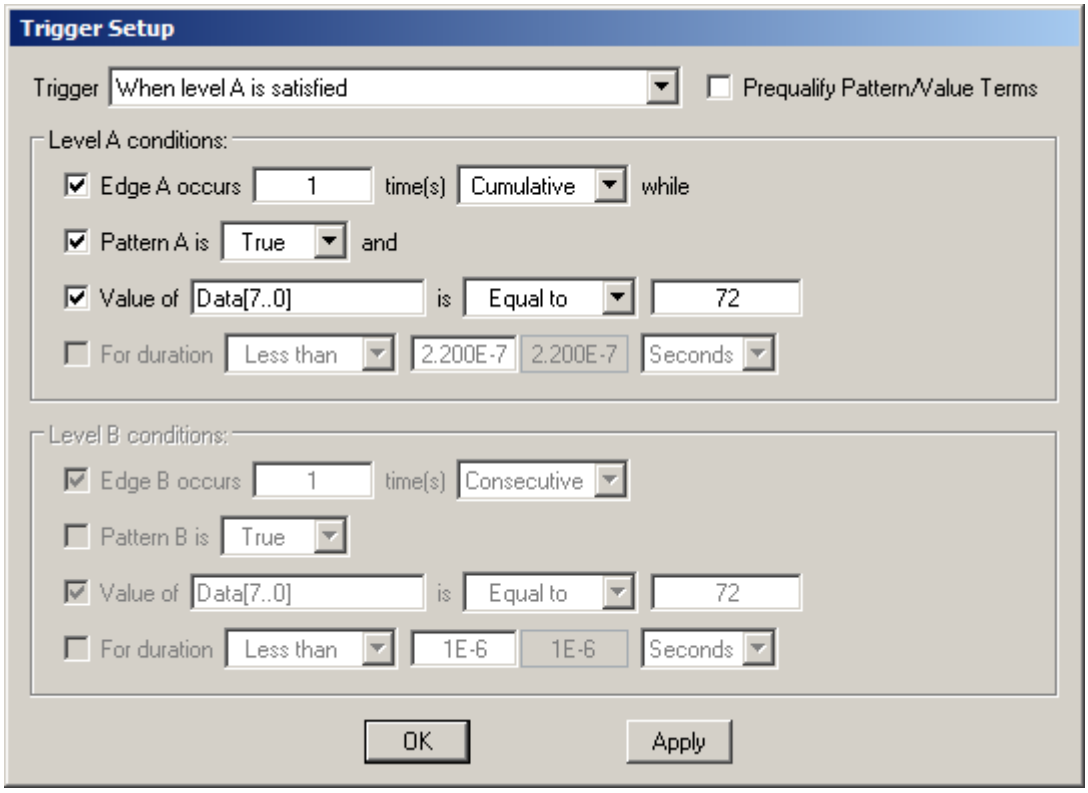
c) Konfigurieren Sie die Signale und Gruppen zur Erfassung entsprechend folgender Abbildung.



Signal	Wire ID	Wire Status	Edge A	Pattern A	Cursor A
E_Clock1	CLK1	H		X	0
RS_Clock2	CLK2	L		X	1
- Data[7..0]					F
Data0	D0	H		X	0
Data1	D1	L		X	1
Data2	D2	L		X	1
Data3	D3	L		X	0
Data4	D4	L		X	0
Data5	D5	L		X	0
Data6	D6	L		X	1
Data7	D7	L		X	0
E	D8	H	↴	X	0
RW	D9	H		X	0
RS	D10	L		1	1

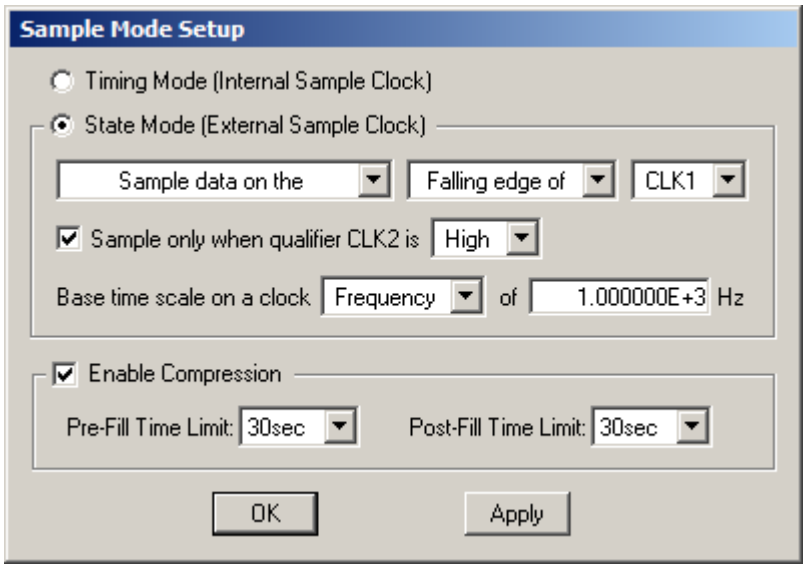
Stellen Sie die Sample Rate auf 200MHz und die Logic Threshold auf 2,5V. Aktivieren Sie den Timing Mode und stellen Sie als Triggerbedingung einen Schreibzugriff mit dem Zeichen 'H' (48h) ein.





Starten Sie eine Aufzeichnung und kontrollieren Sie die Einhaltung des im Datenblatt spezifizierten Timings für einen Schreibzugriff auf das Data-Register.

d) Deaktivieren Sie den Trigger und wechseln Sie in den State Mode und starten Sie eine neue Aufzeichnung.



e) Ergänzen Sie Ihre Testanwendung zur Ansteuerung der IIC-Peripherie. Nutzen Sie dazu die gegebenen Funktionen aus dem Arbeitsmaterial zur VL Mikrocontrollerpraxis zum Schreiben und Lesen eines Bytes über den IIC-Bus sowie zur Initalisierung folgende Funktion.

```
void init_i2c(void)
{
  SSPCON1bits.SSPM3 = 1;
  SSPADD = 49;     //SSP Baud Rate Reload Register in I2C Master Mode.
  SSPCON1bits.SSPEN = 1;
}
```

Ergänzen Sie die folgenden Symbole zur Adressierung, sowie die Endlosschleife der Funktion main.

```
#define WRITE_LED_7SEGMENT 0x40 // Adr. 0x20 + R/W-Bit = 0
#define READ_SWITCH 0x4F        // Adr. 0x27 + R/W-Bit = 1
write_i2c(WRITE_LED_7SEGMENT, read_i2c(READ_SWITCH));
```

f) Ergänzen Sie in der Signalkonfiguration der Software LogicPort entsprechend folgender Abbildung einen I2C-Interpreter.